Spring 2020

Piezoelectric Energy Harvester Improvement

Nathan Embaugh
nle10@zips.uakron.edu

Jason Mack
jpm107@zips.uakron.edu

Jeremiah Fitzgerald
jsf42@zips.uakron.edu

Zachary J. Lindsey
zjl11@zips.uakron.edu

Piezoelectric Energy Harvester Improvements

No. 2, Project Number B14

04/28/2020

Authors:

Nathan Embaugh

Jeremiah Fitzgerald

Zachary Lindsey

Jason Mack

The University of Akron

4600:471:001

Advisor: Dr. Siamak Farhad

Graduate Student Advisor: Roja Esmaeeli

Graduate Student Advisor: Haniph Aliniagerdroudbari

Abstract

This report discussed the progress and improvements of the design, manufacturing and testing of a cymbal style piezoelectric energy harvester [PEH] made for usage in an automobile tire. Background information and previous work on this initiative were discussed briefly. After this, tasks performed by the group this past academic year were highlighted. A new simple way to assemble the PEH was mentioned along with a short COMSOL study. Tasks including the improvements and overhaul of a measurement system for data acquisition were discussed next. Instructions and documentation on how to utilize the measurement system were provided in this report. An enclosure for the measurement system was also created as another testing improvement. Plans for future testing were outlined. Research objections throughout the year were summarized and finally more documentation was provided on the Arduino and measurement system. Impactful improvements along with detailed documentation will allow for immediate success for future groups.

Table of Contents:

1 - Introduction and Background

1.1 - Introduction:

As autonomous vehicles are increasing in demand and use, the need for more sensors on the car to provide more data has also increased. Tire sensors can be used to measure tire and road conditions. The problem of powering such a tire sensor becomes an issue due to the fact the sensor is located on the inner liner of the tire. Attaching the sensor to an external power source via wires is impractical in this scenario as the tire is rotating. Therefore, it would be beneficial to have an energy source located within the inner liner as well. The use of piezoelectric energy harvesting systems have been shown to be an effective solution to this problem.

The work of design and manufacturing was completed previously, this report will detail a continuation of improvements to the system. An improvement to the sampling rate was of the highest priority. Other tasks include a housing environment, manufacturing fixtures, adding sensors, and other improvements for testing were made.

1.1.1 - Conceptual Design:

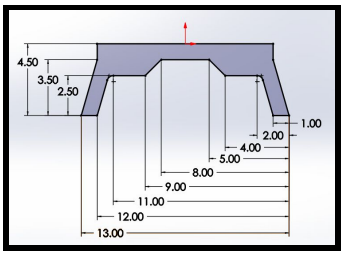
The conceptual design phase in regards to this report was conducted previously by the prior group. The following background information on energy harvester technology and cymbal energy harvesters was taken into account.

1.1.2 - Piezoelectric material:

Piezoelectric materials create an electrical voltage from mechanical stress. These materials are found in both nature and industry. Examples include bones, Berlinite, Barium Titanate, and Lead Zirconate Titanate. These materials can be implemented into systems called piezoelectric energy harvesters (PEH). One application of a PEH is displayed here; installed into a flexible metal housing and attached to the inner lining of a car tire. The normally wasted energy can now be harvested and turned into electricity to power sensors on the automobile. The material specifically chosen for this project's application is PZT-5h, this was based on previous research conducted by Roja Esmaeeli [9]. This paper looks into the design, modeling, and the analysis of high performance PEH.

1.1.3 - Cymbal PEH Configuration:

There are several different housing designs for PEH devices based on their application such as Rainbow PEH, Cantilever Beam PEH, and Cymbal PEH. For harvesting the strain energy of the tire, this Cymbal PEH Configuration was selected as it efficiently converts the tension and compression of the tire to longitudinal strain in the piezoelectric material. Other advantages of this design are that the metal end caps improve the endurance of the piezoelectric material under high loads and has a lower manufacturing cost compared to the other systems. The PZT-5h is glued in between the top and bottom pieces of the housing, with wires attached to the electrodes and leading to the sensor system that is mounted on the rim of the wheel. The cymbal pieces are manufactured from a low carbon steel. (Dimensions in mm)



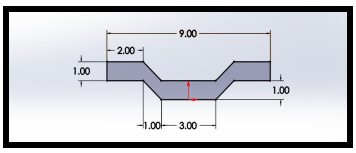
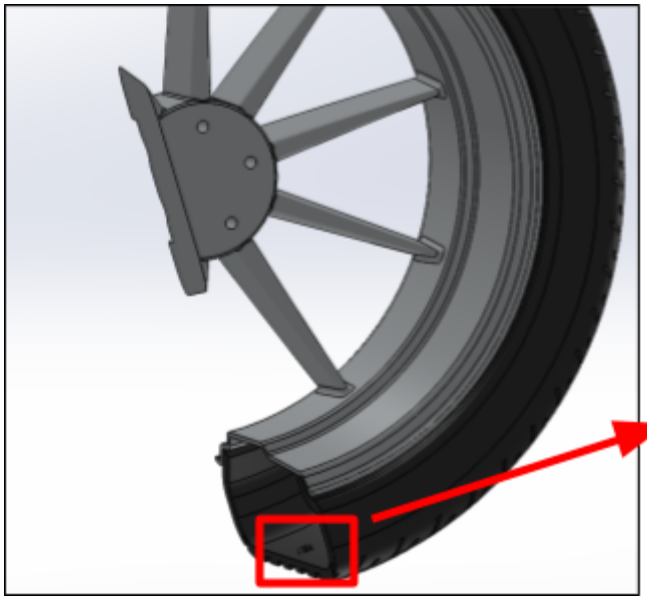


Figure 1: Dimensions of Top Portion of Cymbal PEH

Figure 2: Dimensions of Bottom Portion of Cymbal PEH



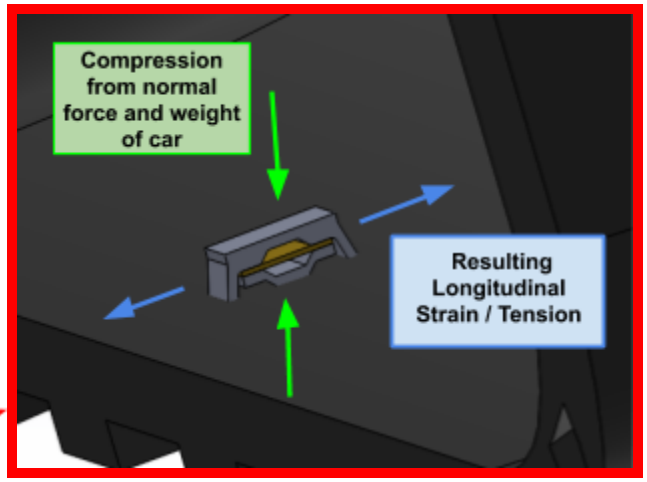


Figure 3: View of Cymbal PEH and Tire Assembly

Figure 4: View of Forces Acting on Cymbal PEH Assembly

1.1.4 - COMSOL Multiphysics Model

In the COMSOL Model developed by graduate student advisors Roja Esmaeeli and Haniph Aliniagerdroudbari, the cymbal PEH configuration is developed to optimize voltage obtained from the piezoelectric material. “The area of energy harvester that is attached to the tire is roughly 0.3% of the tire inner surface area. The basic of strain-based piezoelectric energy

harvesters is to use the longitudinal strain produced in the tire while it is in contact patch as a mechanical input for direct piezoelectric effect. Contact patch is the point that the tire gets in touch with the pavement… The inner surface of the tire becomes under tension. In addition, just after and before the contact, the inner surface of the tire becomes under compression. It is obvious that the tension of the tire inner surface is higher than the compression before and after the contact patch."

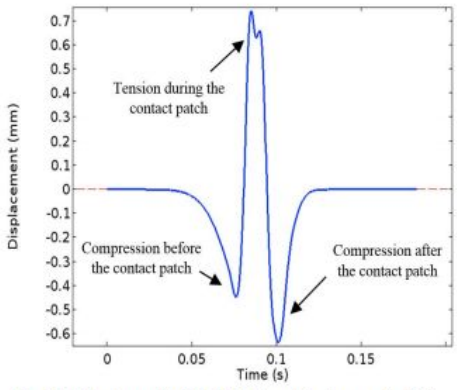
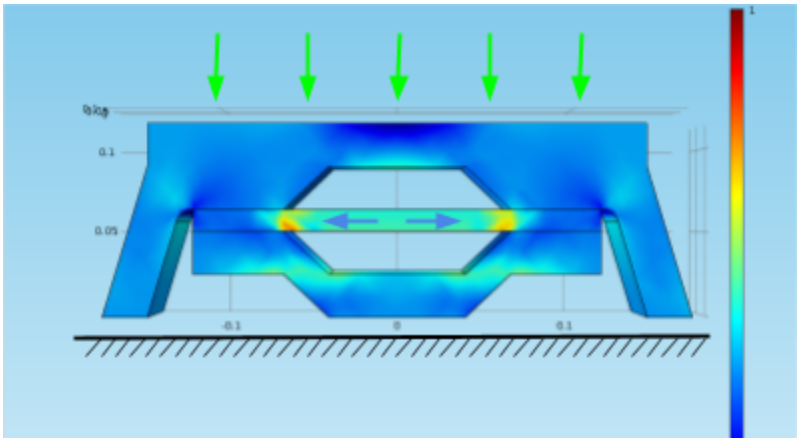


Figure 5: Graph of Tire Deflection over One Rotation at 41 km/h [4]

In this simplified COMSOL Multiphysics model, the deformation of the PEH is displayed. This model was created by group members this semester to display a visual representation of the displacement occurring to the piezoelectric material. With the bottom (Contact Patch) fixed,

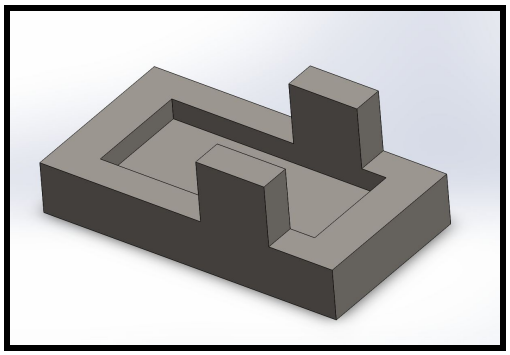


and a load applied to the top, a longitudinal deformation can be seen in the piezoelectric material.

Figure 6: COMSOL Multiphysics Model of PEH Assembly

1.1.5 - PEH Assembly:

Due to the relatively small size of the PEH, assembling the two housing pieces, piezoelectric material, and wires proves to be difficult. A small assembly tool to hold the pieces in place while the glue sets would help with the assembly process. After analyzing several different designs for such a tool, this simple piece can be 3D printed and used to hold each of the three pieces in place while also leaving space for the wires to be attached. Because the upper piece has a flat top, it can first be placed into the assembly tool upside-down, followed by the PZT-5h and bottom piece. These figures show the assembly tool as well as how the pieces can be oriented and placed into it during the assembly.



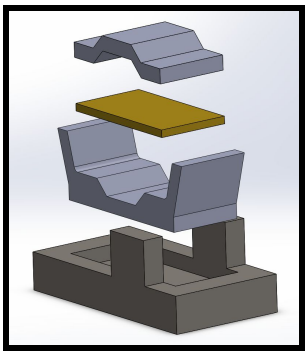
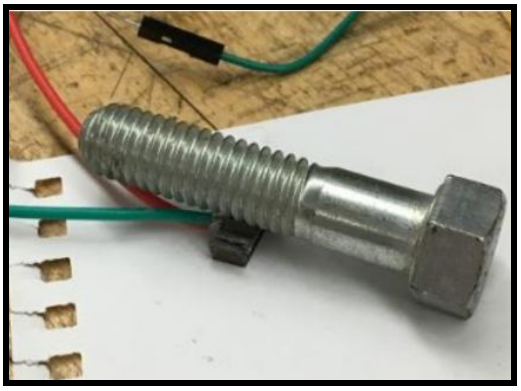


Figure 7: 3D Model of PEH Assembly Tool

Figure 8: Exploded View of PEH Assembly and Tool

Not only does this tool aid in assembly, it can also potentially improve measurement accuracy due to the more precise set up that will more accurately align the pieces. This tool can be easily 3D printed for a low cost yet makes several improvements to the PEH. A small weight

can easily be added to the top to aid with glue setting. Shown below is the previous assembly process compared with a model of the assembly tool and pieces fitting together.



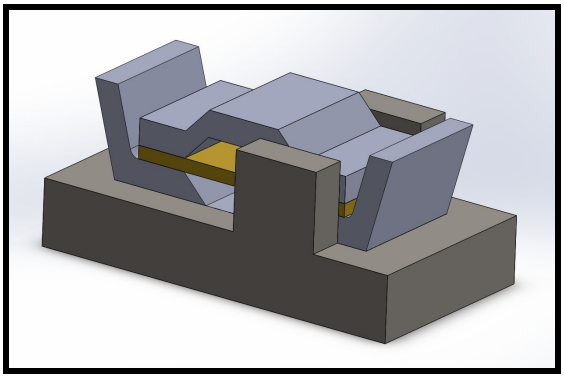


Figure 9: Example of Weight Aiding in Glue Setting [4]

Figure 10: PEH Assembly and Tool

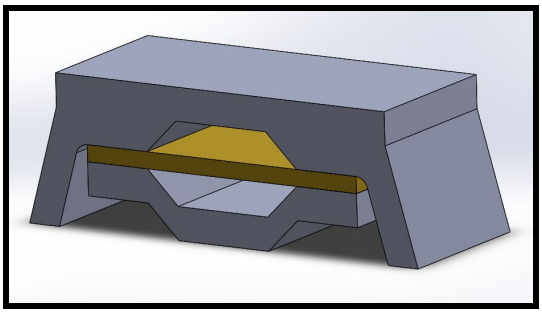


Figure 11: 3D Model of PEH Assembly

2 - Tasks

2.1 - Data Acquisition System

2.1.1 - System Overview

The Data Acquisition System was the central focus of the work done in the project. The requirements of the system were as follows:

1. Wireless
2. Obtain sensor readings at a minimum sample rate of 100 hertz
3. Capable of measuring five inputs at once
4. Lightweight and small

The work done in the year prior had created the base for which the system was to consist of, and simply needed to be improved upon. The system that was inherited consisted of an Arduino MKR1000, a voltage sensor,a current sensor, and accompanying battery. This system was capable of recording sensor data at a sample rate of twelve hertz, and provided the readings of two sensors. The programming of this system used Matlab and its functionality with Arduino systems. A Matlab script was created using the Arduino toolbox that took two hundred readings, with no specified sample rate. The sample rate was determined by how long the code took to run, which was both inaccurate and difficult to work with.

It was determined that using a Matlab script and its built in Arduino functions was not going to be capable of high speed sample rates such as one hundred hertz. In trying to remain under the Matlab umbrella there was an attempt to use Simulink, a subsystem of the Matlab program. Simulink looked promising, giving choices to specify high sample rates, along with being easy to upload to the Arduino and to the computer but presented several problems. The Simulink-Arduino toolbox is rather new and has very little support both within the University and from other projects that have been done around the globe. It was determined that due to the lack of resources for how to use the Simulink-Arduino connection, it would be best to pursue other options.

The hardware of the data acquisition system is the following:

1. Arduino MKR1000
2. 2 Breadboards
3. A LiPo battery
4. Wingoneer Voltage Divider/Sensor
5. Sparkfun Low Current Sensor
6. 3 Strain Gauges (extra hardware possibly necessary)
7. Enclosure (in development)

The voltage and current sensors require extra explanation that will be detailed below in the following section.

2.1.1.1 - Voltage Divider and Current Sensor

To prevent damage to the Arduino board, a voltage divider must be implemented into the set up. Because the COMSOL model of the PEH predicts a voltage greater than 3.3V, a Wingoneer Voltage Divider consisting of two resistors (R1 = 30 KΩ, R2 = 7.5KΩ) must be

implemented. The voltage divider can be represented mathematically as:

$$V_{in} = V_{out}\,(R_2/(R_1 + R_2)\,.$$

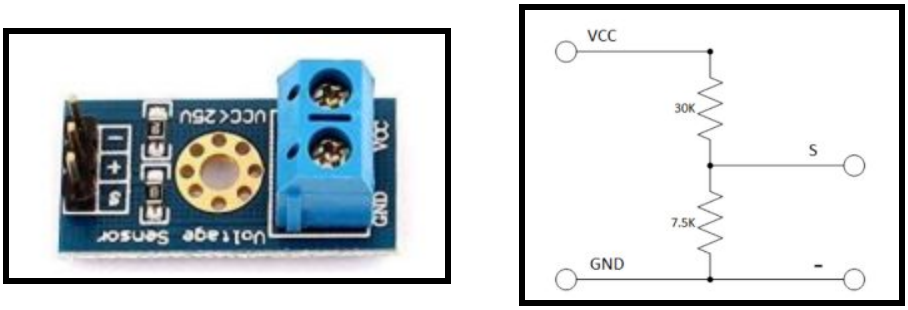


Figure 12: Voltage Divider & Schematic [4]

In order to achieve the goal of measuring the power, a current sensor must also be included in the system. To create a voltage signal, the SparkFun Low Current Sensor Breakout-ACS723 uses Faraday's Law of Induction and the Hall Effect. Further calibration will most likely be necessary, and guides on this calibration can be found on the vendor's website.

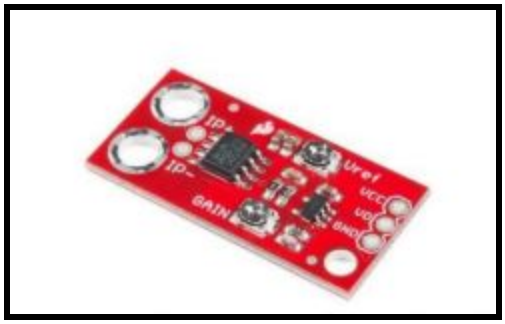


Figure 13: SparkFun Breakout-ACS723 Low Current Sensor [4]

2.1.2 - Programming Implementation

Once it had been determined that other options outside of Matlab would be chosen, the obvious choice was to use the built in Arduino IDE and it's programming language, C++. The

support system of the Arduino community is immense, meaning that any problems that may occur would likely have a resolution already found out, and just needed to be implemented.

The MKR1000 is a board that runs on a 5V power supply generated by a Li-Po single cell, 3.7V, 700 mAh battery. It has a circuit operating voltage of **3.3 Volts**. This meant that all input voltages coming into the MKR1000 must be under 3.3 volts. The MKR1000 has seven analog input pins, able to report in 8, 10, and 12 bit modes. For these purposes, due to the relatively small voltages, the MKR1000 was set to read in 12 bits to provide the greatest resolution. This MKR1000 was programmed using the Arduino IDE and it was then programmed from scratch. There were three main challenges that had to be solved with code within the code uploaded to the MKR1000. The solutions that were implemented for each part will be detailed in the following sections.

2.1.2.1 - Connecting to a Wireless Network

Connecting to a wireless network with an MKR1000 was a relatively simple task. Using the Wifi101 module and examples that were provided from Arduino's website, the MKR1000 was programmed to connect to a wireless network with a security type of WPA2. The ssid and the password of the wireless network are all that is required.

```
10 #include <SPI.h>
11 #include <WiFi101.h>
12 //Wifi Connection
13 char ssid[] = "NathanE" ;
14 char pass[] = "Arduino1" ;
15 int status = WL_IDLE_STATUS;
```

Figure 14: MKR1000 Wifi Parameters

The “ssid[]” is simply set equal to the name of the wireless network and the “pass[]” is set to that wireless network’s password. Once these parameters were input, the MKR1000 was connected to the network and ready to begin sensing once properly programmed.

2.1.2.2 - Sampling at a High Frequency

The system as it was given to this group sampled at a rate of twelve hertz, and had no control as to how it was taking samples at this speed. The solution to this problem was determined to be a simple for loop, that checks if enough time has passed to take another reading. The final testing frequency obtained by the completed system was a rate of two hundred hertz, or every five milliseconds. Once five milliseconds have passed it takes a reading of all five sensors, and then waits for five more milliseconds to pass. This time tracking is done by the built in Arduino function called micros() which tracks how many microseconds have passed since the Arduino has been powered on.

Once this was implemented, it became prudent to understand what the frequency limitations of this new programming method would entail. It became clear that the micros() function would allow sampling speeds of up to ten thousand hertz, but there are other factors that affect the sampling rate. The limiting factor in this implementation is the number of times the “analogRead()” function is called. The analogRead() function is a built-in function of Arduino that does the math necessary to read the voltage coming into a specified input. This function takes roughly 100 microseconds, and in doing this function five times a row it can introduce what is called jitter, or a slow shifting of the sample rate, so that it isn’t taking readings exactly when it should, but instead several hundred microseconds later. This jitter is less impactful at speeds of around two hundred hertz, but may become impactful moving up in higher speed sampling rates.There are ways of avoiding the analogRead() function that may allow even higher sample

rates, but these methods were beyond the scope of this year's work. Thus the MKR1000 was programmed to take a reading every 5000 microseconds (5 milliseconds), or at 200 hertz.

2.1.2.3 - Sending Data Wirelessly

The next large challenge was to take that large amount of data, and send it over the wireless network to a computer connected to that same network. The downside to working outside of Matlab is that this was not built in, and had to be coded in from scratch. Using several resources that have done similar projects, it was determined that by using HTTP communications over the wireless network, the MKR1000 could communicate with a XAMPP server that is running on the computer that is on the same network. There were several components to this connection to make it possible:

1. The MKR1000 needed to send its data (MKR1000)
2. The server needed to receive the data (Local Computer, XAMPP Server)
3. A way to store the data (Local Computer, XAMPP Server)

Upon first glance, the components above may seem overwhelming, but due to the large amount of community support for these platforms, much of it was already done and only needed to be adjusted for our purposes. An in-depth explanation of how to use and set these things up will be detailed in the "How to Proceed" section of this report.

2.1.2.3a - MKR1000 Sending the Data

Using HTTP communications the Arduino is able to connect to any given server. The MKR1000 was set up as a client, and then connected to the server that is running on the local computer. Once the connection is made, the MKR1000 is capable of using HTTP communication such as the GET method. The GET method is capable of requesting data from a

server, as well as sending data to a server over any network, wireless or wired. Below is an example of a statement that the MKR1000 would send:

“GET /TireSensing/dht.php?time=4.751&voltage=0.0363&current=0.0000&strain1=0.0000&strain2=0.0000&strain3=0.0000 HTTP/1.1”

This data is formatted in a way that the server will be able to read and handle the information, detailed in the next section.

2.1.2.3b - Receiving the Data

Once the MKR1000 has sent the data, it must be properly received by the server that is running on the local computer or laptop. First the web server must be created, for this project this was done through XAMPP, a free and open source web server solution. Once XAMPP is set up on the computer, a PHP script must be created that can read the GET commands being sent from the MKR1000 and do the proper processes with the information that is sent. The PHP script will read each **“&variableName=0.0363”** and then it must take the value that it reads and put it into a database.

2.1.2.4 - MySQL Database

The final step of the sensing and storing process is storing the sensor data in an accessible way. Built into XAMPP is a module named MySQL which is an open source SQL database management system. The PHP script is programmed to connect to a specific database and add lines of data to each variable name that is specified within the PHP script. Once the database and PHP script are coordinated to work together, the process is set up to receive and store data wirelessly. From the MySQL database management system, there are several ways of exporting the data for post processing and analysis.

2.1.3 - Strain Gauge Implementation

At the beginning of this project it was requested that three strain gauges would be implemented in addition to the current and voltage sensors. Three strain gauges were chosen to determine strain in the horizontal, vertical, and diagonal directions of the tire when looking at it from the center of the tire. Strain gauges are compatible with the Arduino MKR1000 but require a voltage source, and an amplifier to make the voltage change more detectable by the MKR1000 analog input. The process of integrating strain gauges had been started, but due to COVID-19 it was put on halt and no significant progress was made. As a starting place for any continuing groups, some helpful resources have been found.

In a paper titled “A study of strain and deformation measurement using the Arduino microcontroller and strain gauges devices”, An explanation of strain gauges interacting with Arduino is explained in detail. This study explores the set up for a uniaxial set up as well as a triaxial rosette. The triaxial set up can obtain the desired horizontal, vertical, and diagonal strain measurements. The study then goes on to detail a circuit used to allow a good sensitivity to the measuring instrument, the Wheatstone Bridge circuit. A quarter, half, or full bridge setup can be used. Because this study then uses these strain gauges for a different experiment, more research and testing will need to occur to determine which form of the Wheatstone bridge circuit will need to be used for the PEH. It is also likely that an amplifier will be necessary.

2.1.4 - Data Acquisition Conclusion

The new data acquisition system consists of 3 physical hardware pieces, an Arduino MKR1000, A wireless router, and a Windows PC. The programming required resides in two

scripts, an Arduino .ino file that runs the sensing and sends the data, and a PHP script that receives the data from the MKR1000 and then reads it into a database. The software requirements of this sensing system is the Arduino IDE and the XAMPP web server program that run on Windows computers.

The final data acquisition system is a system that is capable of reading five sensors at a rate of 200 Hz. This sensing rate will allow for well defined curves of all sensors when used in an automotive application at normal roadspeeds within a tire.

2.2 - Enclosure Creation

With the creation of this measurement system there will need to be a series of tests done. To test the system, it must be secured to the inner liner of the tire. In previous tests, the measurement system components were simply taped to the tire using duct tape. It was decided that was an unreliable testing setup and an enclosure for the measurement system and an improved method of securing it was needed.

For the enclosure design, it was required that it would be able to hold two breadboards, an Arduino MKR1000, and a battery while also having space for wire connections. Also, it was needed to have a way of sensors and wires to leave the enclosure to be connected to the inner liner of the tire. A 3D model of the system was created and then an enclosure was designed around it to ensure its fitment. One feature is the enclosure utilizes the breadboard's snap on feature to secure the breadboard and eliminate any play during testing. Slots along the front and side of the enclosure allow the wires to connect the sensors to the breadboards.

The next requirement is a method of securing this enclosure to the tire and wheel assembly. The enclosure is designed to be mounted to the wheel itself. This is accomplished

by including “belt loops” to the base of the enclosure and then by wrapping hose clamps around the wheel and looping through the belt loops, the enclosure is secure for testing.

The final main feature is a lid to close the enclosure, that will completely secure the measurement system. The lid was simply made to match the top cross sectional area of the enclosure. To secure the lid a circular boss feature was added to each corner of the housing and lid. Then using a self tapping plastic screw the lid can be taken on and off.

For testing purposes it was decided that the enclosure system will be 3D printed. One reasoning for this choice was the savings in cost and the turnaround time. It was known that there may need to be a few versions of this design so keeping the cost low and having the quicker turnaround time for parts while having deadlines was a must.

There are some noticeable improvements that can be added before future testing. The first improvement would be to change the flat base to a curved base that fits the radius of the wheel. Also, cleaning up the inside of the housing may be beneficial and possibly save on material and weight. Another notable improvement will be any fine tuning of the enclosure. This will most likely be needed due to the outcomes of the international pandemic.

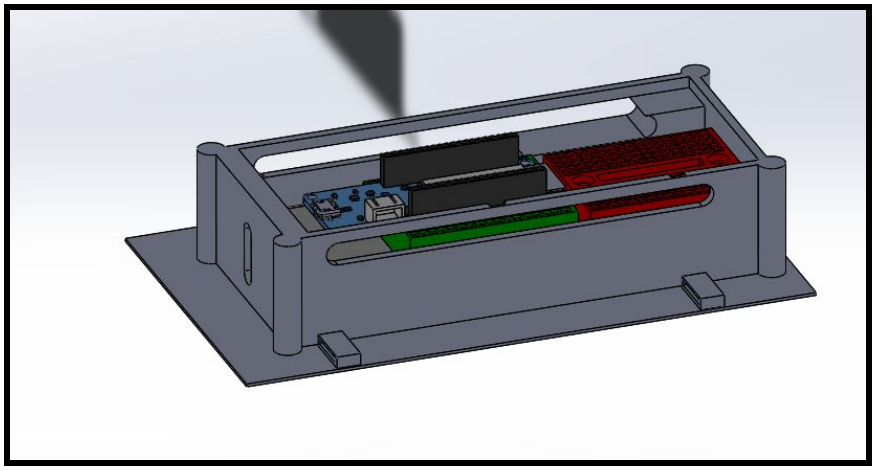


Figure 15: Isometric view of enclosure

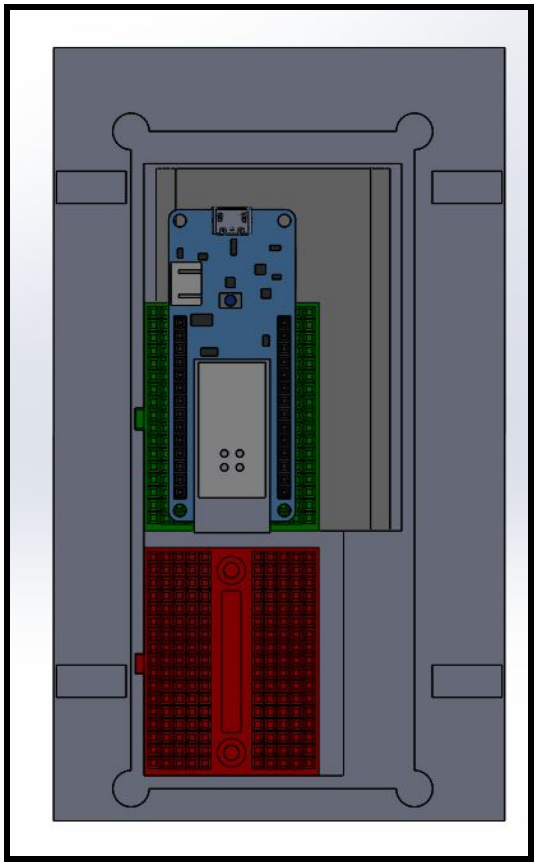


Figure 16: Top down view of enclosure

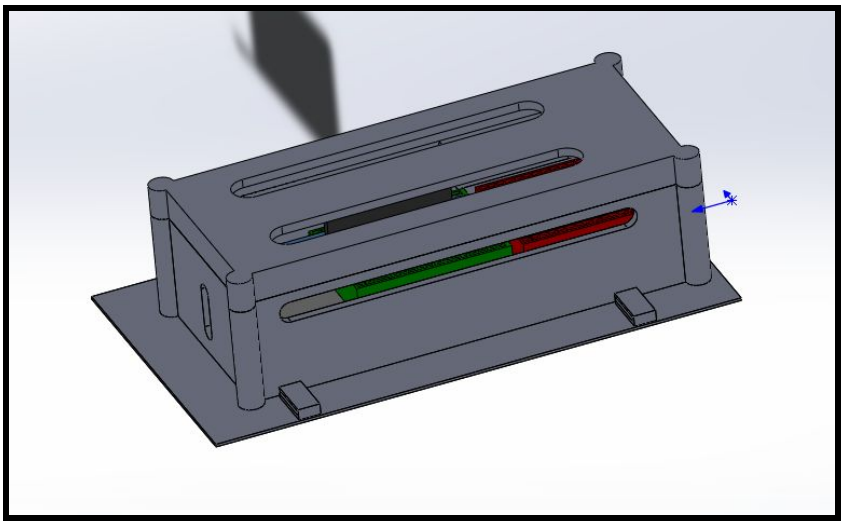


Figure 17: Isometric view of enclosure with lid

2.3 - Future Testing & Methodology

2.3.1 - Stark State Contact Information

Throughout the duration of this project and research from the University of Akron, Stark State College Automotive Technology Center was used as a testing facility for verifying data. This year's group visited the facility and reconnected with staff. The contact information for these individuals is listed below.

Name	Title	Phone	Email
Michael Conway	Department Chair	330-494-6170 ext. 4936	dconway@starkstate.edu
Samuel Adair	Program Coordinator / Associate Professor	330-494-6170 ext. 4728	adair@starkstate.edu
Brandon Diana	Lab Assistant	330-494-6170 ext. 5236	bdiana@starkstate.edu

Figure 18: Contact information for Stark State College Automotive Center

When working with Stark State, it is important to reach out 2 - 4 weeks before the desired testing period. It will also be necessary to bring the Rim that was purchased this year, along with the measurement system and all necessary accessories to execute the data collection process.

2.3.2 - New Test Methodology

Another improvement that was accomplished this academic year was the establishment of a new test method for verifying data. The previous group had tapped the measurement

system to the inner liner of the tire, along with the wires from the measurement system to the PEH. A picture of this can be seen in the figure below.

Figure 19: Previous Test Method [4]

There are several issues with this method of testing. One problem is that it is very easy for the tape that is holding the measurement system down to come undone which then will force the measurement system to bounce around the inside of the tire during rotations. When the measurement system is unstable it can become damaged by the hitting the rim or create problems with the wiring, or bump the PEH causing that to become loose.

After analyzing the current testing and looking for areas of improvement, a new test method was developed. A figure can be seen below demonstrating this.

Figure 20: New Test Method

In the figure, two hose clamps are screwed around the outside of a tire. The hose clamps will be oriented through the notches on the enclosure for the measurement system, and then tightened so that the measurement system is secure and safe. This will ensure that the measurement system does not become loose and bounce all over the tire. A similar methodology is demonstrated in the figure below where a GoPro was mounted to a rim so that footage of the inside of a tire could be obtained during testing.

Figure 21: GoPro mounted on Rim using hose clamps

The hose clamps and the enclosure will hold the measurement system in place. Wires will freely flow out of the enclosure to the PEH. Future improvements to this system will include more modeling of the enclosure so that the base of the enclosure matches the radius curve of the rim. After working with Stark State on site, it was deemed that a rim needed to be acquired. A rim was purchased and is shown in the figure below.

Figure 22: Rim acquired for testing [7]

This rim is a American Racing AR904 Satin Black Wheel (16x7”/5x112mm, +40mm offset). This rim was picked for several reasons. The best feature of this wheel is that the cross section is relatively flat. Almost all OEM and replacement wheels have a complicated cross section, with many turns and divets. This makes modeling an enclosure base on the cross section very difficult. A figure of this is demonstrated below.

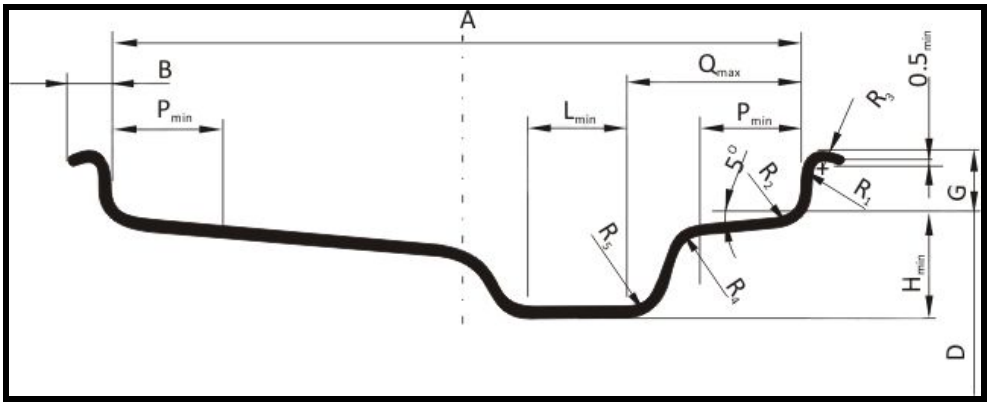


Figure 23: Cross section of typical wheel [8]

A relatively flat/slightly curved cross section is desired so that the enclosure can rest. The rim that was purchased has a flat cross section and was fairly inexpensive compared to other options. Future work on testing will require the group to make improvements to the enclosure by modeling the base of the enclosure to the curve of the radius of the rim that was purchased. A tire will also be needed for the testing. The rim that was purchased works well with the tire size 205/55R16. The tire size can be found on the sidewall of the tire, a figure below shows this information.

Figure 24: Tire Size on Sidewall

Other 16 inch tires also can be used with the purchased rim. It was deemed not necessary to acquire a tire because Stark State has many tires. Future testing will require

coordination between the group and Stark State to select an appropriate sized tire for the rim. Testing with the same tire would be ideal for continuity. It will also be important to cautiously mount the tire onto the rim with the hose clamps and measurement system fastened. The technician and group will have to mount the opposite side of the tire first and then slowly rotate around the wheel. A figure shown below demonstrates this process.

Figure 25: Mounting the tire over the rim [4]

Another initiative that required investigation was finding a new glue for the piezoelectric material. This was seen as an area where improvements could be made to the system. A non-conductive glue that works specifically with piezoelectric material was desired. Boston piezoelectric group (BPG) was reached out to because of their initial involvement with this project through supplying the Piezoelectric material. BPG connected the group to a supplier in the New England area that specializes in these types of non conductive adhesives. After working with their sales team and identifying needs, two products were discussed for purchase. The group received quotes, but just for small samples the cost was almost $300, which was very expensive. Moving forward, it was suggested that future groups ask if the glue could be sampled for a discount or for free since students are working on a project. Another suggestion

was to reach out to contacts at the University of Akron. Several outreach attempts were made to the Polymer college, but no samples were able to be provided. Improvements to glue and taping of the piezoelectric material to the assembly component, and inner liner of the tire could be further investigated moving forward with future work.

Lastly, the previous group was able to run tests at 30 and 35 psi along with forces of 812.5 lbs. Future testing should include testing at speeds over 25 mph, ideally highway speeds with a force of 1000 lbs. These testing conditions will more closely simulate the experience the conditions a tire faces in a midsize sedan vehicle. These desired testing conditions should be able to be reproduced in the Hunter Road Force Tire Balancer that Stark State uses for testing. This summarizes the new testing methodology and general testing improvements that were accomplished this past year. Utilizing this new method of testing could not be accomplished due to a global pandemic.

2.4 - Project Expenses

The figure below is a summarization of the associated expenses of this project this year.

Item	Cost	Purchased From
Comidex 3PC BF120-3AA 120 Ohm High Precision Resistance Strain Gauge	$6.29	Amazon
PZT-5A compressed Crystals Fine Lapped Finish 0.354" Long x 0.197" Wide 0.020" Thick Chrome/gold electrodes both sides Quantity: 10	$500.00	Boston Piezo-Optics Kim Simmons Sales Manager Tel: 508-966-4988
American Racing AR904 Rim Satin Black Wheel 16x7"/5x112mm +40mm offset	$97.44	Amazon
Total Cost		**$603.73**

Figure 26: 2019 - 2020 Project Expenses

Future expenses may involve purchasing hose clamps for testing. These are very cheap and can be acquired at any hardware store. Additionally there may be purchases necessary for strain gauge implementation. A special thank you to the Industry Council of the Mechanical Engineering Department for financial support throughout the year.

3 - Research & Expected Outputs

Piezoelectric energy harvesters are relatively new systems that have great potential and room for improvement. That being said, several systems similar to ours have been tested before as well as some different configurations. Research was crucial to understand the kind of data to expect and which configurations give the best outputs, especially due to the inability to test the PEH detailed in this report.

Research showed an article titled, “High-Performance Piezoelectric Energy Harvesters and Their Applications”, where one such application involved a tire pressure monitoring system. This system used a PEH with a cantilever configuration, as opposed to the cymbal configuration outlined in this report. “When mounted on the tread wall and tested on the road at ~70 km hr−1 driving speed, the prototype generated a power of ~30 μW on average.” Testing planned for our PEH is similar, but utilizes the Roadforce Elite Wheel Balancer so tests can be repeated in a more controlled environment. This data is most closely related to the PEH used in a similar application to ours, but taking into account several other experiments, the researched summarized that “In summary, the power output of the developed energy harvesters ranges from10 μW to 1 mW in on-road tests.” This gives a great range of power outputs to expect once testing occurs.

Another resource used to obtain a range of expected outputs for the PEH comes from a paper titled “Energy Harvesting Technologies for Tire Pressure Monitoring Systems”. In this paper, similar results are observed from cantilever beam, tire mounted PEH systems. Powers of 47μW and 65μW were reached using the cantilever system. More data is given from exploring other configurations of PEH systems. Even higher power outputs were achieved through the use of a “bender” configuration, similar to the setup outlined in this report. “The power generated was stored into a capacitor and relatively large power levels (6.5 mW) where achieved.” This high power output is a good indicator that this type of configuration does well at maximizing the voltage created by the piezoelectric material. The study also goes on to provide exciting data from tests involving several sheets of piezoelectric material. “Scale up of the use of PZT bender harvested was examined by using a large 4 × 40 array of benders on the inner surface of the tire.52 The voltage was rectified and stored in a capacitor. A large power of 2.3 W was produced at a speed equivalent to 100 km h−1, which was doubled to 4.6 W using two layers of devices.” Also, due to the fact the topic of this report is a new technology, there are no industry standards exactly regarding this system. Therefore, standards on tire pressure monitoring systems for midsize cars were referenced.

4 - Data Acquisition System Guide

As described in Section 2.1, there are several components to the new sensing system, that require a certain amount of configuration and setup. As this project will be continued in following years after this group, it became important to document the processes that would be required to continue this project with minimal downtime. This section of the report will detail the set-up and use of the following systems:

1. **Arduino MKR1000 and its code**
2. **XAMPP Server and its PHP script**
3. **MySQL Database and its setup**

All programming files will be included in the appendix of this report, and will also be left with Dr. Farhad and his research assistants. These files will be configured in a way so that it will be a quick setup with minimal configuration, for a sensing system of five sensors at 200 Hz. The sensors will be as such:

Voltage - Connected to A0 input

Current - Connected to A1 input

Strain1 - Connected to A2 input

Strain2 - Connected to A3 input

Strain3 Connected to A4 input

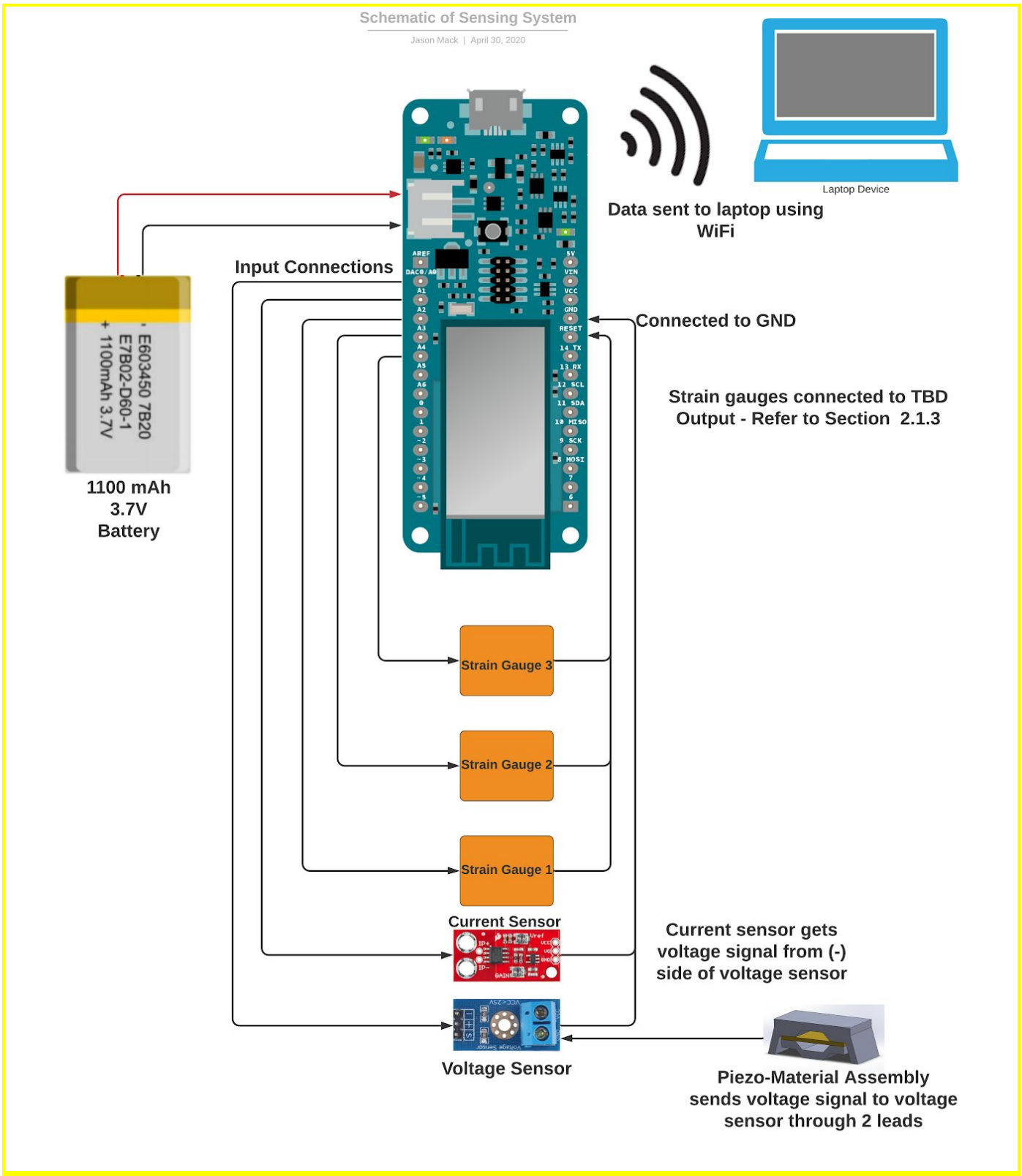


Figure 27: Wiring Schematic of Existing Setup

If configuration changes are needed, this section will explain the function and purpose of the most important parts that might need changed.

4.1 - Arduino MKR1000 and its Code

The Arduino MKR1000 is a wireless enabled microprocessor that is easily used in applications such as this. All Arduino's are coded using the Arduino IDE, which is a coding program that is based in the C++ programming language. The Arduino IDE download may be found here. Note: it is recommended to use the "Arduino IDE" and not the web editor. Once the Arduino IDE is installed, the "Wifi101" library will need installed which can be found in "Tools>Manage Libraries…" and the "Atmel SAMD Core" must be installed which can be found in "Tools>Board:>Boards Manager…".

All Arduino scripts have three sections, declaration of variables, a setup(), and a loop(). The setup() runs one time upon the Arduino powering on, and the loop() runs continuously until the Arduino is powered off. Below each section of the code will be broken down, including important notes of what and why certain lines have been written.

4.1.1 - Declaration of Variables

```
//Wifi Connection
char ssid[] = "NathanE" ;
char pass[] = "Arduino1" ;
int status = WL_IDLE_STATUS;

//Setting up client parameters
IPAddress server(192, 168, 0, 103) ;
WiFiClient client;
```

Figure 28: Wifi Variables

The first section of the Arduino script is shown above, these parts will almost certainly need modification to work properly with your computer and router. First, ssid[] must be set equal to the name of your wireless network, in the above example the wifi name is NathanE without quotation marks, the quotation marks indicate to the software that it should read the variable as letters. “Pass[]” must be set equal to the password of the wireless network, in the above example the password is Arduino1. Lastly, the IP address of the laptop computer that will run the XAMPP server must be put in IPAddress server(###, ###, ###, ###), in the above example the IP address of the server is “192.168.0.103”.

```
 //Time Variables
float timeSent ;
int interval = 5000 ; //Time between sensor readings in microseconds
int sampleSize = 1000 ; //Number of readings taken per sample
unsigned long timestamp [1000]; //Needs to be as big as the sample size
unsigned long previousTime = 0 ; //Used for determining when to take a reading
unsigned long currentTime = 0 ; //Used for recording time to send to database
unsigned long initialTime = 0 ; //Used to make each set of readings start at zero seconds
```

Figure 29: Time Variable Declarations

This set of time variables defines the sampling rate by specifying how much time to pass between each reading in the “interval” variable. In this instance, 5000 microseconds is the value chosen, or 5 milliseconds. This equates to a sampling rate of 200 Hz. It is not recommended to change the interval to any lower number, as 200 Hz was determined to be the limit to this sensing systems capability. The sampleSize variable determines how many readings it should take at the sampling rate specified. It was determined that 1000 samples is roughly the largest sample size that can be taken before the MKR1000 runs out of internal memory. If sampleSize is changed, all of the variables with [1000] must be updated to match whatever the new sampleSize value is.

```
//Sensor Variables
float voltage [1000]; // Array to hold voltage values, needs to be as big as sample size
float current [1000]; // Array to hold current values, needs to be as big as sample size
float strain1 [1000]; // Array to hold strain1 values, needs to be as big as sample size
float strain2 [1000]; // Array to hold strain2 values, needs to be as big as sample size
float strain3 [1000]; // Array to hold strain3 values, needs to be as big as sample size
```

Figure 30: Sensor Variable Declarations

It is recommended to not change these variables, unless the sample size is changed, in which case all the numbers within brackets should match that.

```
//Conversion Variables
float sensorToVoltage = (1.0 / (7.5 / (7.5 + 30.0)) * (3.3 / 4095.0)) ; //Used to convert from digital to analog value
float sensorToCurrent = (3.3/4096.0) ; //Used to convert from digital to analog value
float sensorToStrain1 = (3.3/4096.0) ; //Used to convert from digital to analog value
float sensorToStrain2 = (3.3/4096.0) ; //Used to convert from digital to analog value
float sensorToStrain3 = (3.3/4096.0) ; //Used to convert from digital to analog value
```

Figure 31: Sensor Conversion Variables

When sensors report back to the Arduino, it comes in a digital form based on the range of voltage that can be reported, and the range of values that the sensor can return. In this case, it is sensing with 12 bits, meaning that analogRead() can return 0-4095 as it's value in which 0 is zero volts, and 4095 is the highest voltage value that can be returned. Due to this, each sensor reading once taken must be converted from the 0-4095 range to the corresponding voltage/current/strain value that is being generated. These variables handle the entire conversion from digital to analog and may be modified depending on which sensor is used. For sensorToVoltage it is a combination of the conversion that must happen due to the voltage divider that is explained earlier in the report as well as the conversion from digital to analog. The current sensor is slightly different in that it also needs to convert the voltage to current which is the (100.0/500.0). This may need modification as testing and calibration is performed that was not able to be done this year due to COVID-19. The final three values are subject to change based on the research done on the strain gauge setup as discussed in Section 2.1.3.

4.1.2 - Setup()

There is nothing that should be modified in this section, its purpose is to connect to the wifi, and confirm that the MKR1000 is able to connect to the server. It will report all of these results via the “Serial Monitor” that can be found in the Tools of the Arduino IDE. This serial monitor is meant to be used when the MKR1000 is connected directly to the PC via USB cable. The only possible thing to modify here would be the very last line, “delay(300000) ;”. This line tells the MKR1000 to wait 5 minutes before it starts the main loop() that will sense and send data to the database. This can be modified to whatever time is necessary, with an understanding that the number is in milliseconds.

4.1.3 - loop()

The loop() is very simple, as stated earlier, it will loop forever until the end of time repeating the commands that are within this. It has three functions, the first two, read_sensors() and Sending_To_Database() are other functions that will be explained later. The final line is delay(30000) which will tell the system to wait 30 seconds until it reads the sensors again. This can be modified if it is desired to have a lesser or greater delay in between sensor readings.

4.1.4 - read_sensors()

```
void read_sensors()
{

  analogReadResolution(12) ;
  if (client.connect(server, 80)) //ensures that the server is connected before taking readings
  {
    Serial.println("Server on, reading sensors") ;
    for (int i = 0 ; i < sampleSize;)
    {

      currentTime = micros() ;


      if (i == 0)
      {
        initialTime = currentTime ; //Store the time at which the first measurement is taken,
        //used to make time start at zero in data processing
      }

      if (currentTime - previousTime >= interval) //Used to take readings at the interval, sets the sampling rate
      {
        timestamp[i] = currentTime ;
        previousTime = currentTime ;

        voltage[i] = analogRead(A0) ;
        current[i] = analogRead(A1) ;
        strain1[i] = analogRead(A2) ;
        strain2[i] = analogRead(A3) ;
        strain3[i] = analogRead(A4) ;

        i = i + 1 ;
      }
    }
  }
  else
  {
  Serial.println("Server turned off") ;
  }
}
```

Figure 32: read_sensors() Code

This function handles the reading of sensors at the specified time interval (sampling rate). First the MKR1000 is configured to use its 12 bit sensing mode that will give a higher resolution, it is not recommended to change this as it is important to have high resolution and all conversions would need adjustment. The MKR1000 then checks to ensure that the server is on, this will be used to control when sensor data is sent to the database. If it is on, it runs through the process of checking if the time that has passed is larger than the interval time, if it is, it

records the time and takes down all five readings. This process is repeated for as many times as the sample size variable specifies. Nothing should need to be modified in this code, it will self adjust to any changes made in the variable declaration section.

4.1.5 - Sending_To_Database()

```
void Sending_To_Database()   //CONNECTING WITH MYSQL
{
  Serial.println("In send to db") ;
  if (client.connect(server, 80))
  {
    Serial.print("Sending to database") ;
    for (int z = 0 ; z < sampleSize ; z++)
    {
      //Reestablish connection to database
      client.connect(server, 80) ;

      //Do Data modification, includes converting from digital sensor values to actual voltage, current, and strain
      //as well as setting time to start at zero

      voltage[z] = voltage[z] * sensorToVoltage ;
      current[z] = voltage[z] * sensorToCurrent ;
      strain1[z] = voltage[z] * sensorToStrain1 ;
      strain2[z] = voltage[z] * sensorToStrain2 ;
      strain3[z] = voltage[z] * sensorToStrain3 ;

      timeSent = (timestamp[z] - initialTime) / 1000000.0 ; //Sets starting time to zero, converts from microseconds to seconds

      //Create the message that gets sent to the SensorScript.php script
      message = "GET /TireSensing/SensorScript.php?time=" + String(timeSent, 3) + "&voltage=" + String(voltage[z], 4) +
                "&current=" + String(current[z], 4) + "&strain1=" + String(strain1[z], 4) + "&strain2=" +
                String(strain2[z], 4) + "&strain3=" + String(strain3[z], 4) +
                " HTTP/1.1\r\n" + ("Host: 192.168.0.101\r\n") + ("Connection: close\r\n\r\n") ;

      //Send the message
      client.print(message) ;

      //Used with Serial Monitor to debug if necessary
      Serial.println(message) ;
```

Figure 33: Sending To Database Code

This function handles data processing and then sending the data to the server and its database. First it checks to verify that it is connected to the server, if it is it then enters the for loop to send the entirety of the sample size. First the 0-4095 sensor values are modified to be it's individual analog value, by the sensorToX variables which were covered in section 4.1.1. The next step modifies the timesent to start at zero. Without this, the time returned would simply be the time since the MKR1000 had powered on, and it would be difficult to tell the difference

between samples taken. Next the “message” is created. This is the most important part of the actual communication between the Arduino and the server. It is highly discouraged to make any modifications to this section of code. It is important to note that the five variable names that are transferred over are defined in this, as all of the words that precede an equal sign. In the above figure it would be “voltage”,”current”,”strain1”,”strain2”, and “strain3”. These variable names will match the variables defined in the PHP script that will be explained in Section 4.2.

4.1.6 - printWiFiStatus()

This function is only used for connecting the MKR1000 to the wireless network, and has no impact on the actual sensing function of the system.

4.2 - XAMPP Server and the PHP Script

4.2.1 - Installing XAMPP

XAMPP is a free open source program that can be downloaded and installed on any computer, but it is recommended to use a Windows PC. The program can be downloaded at this link. Through the download process, ensure that all permissions are given so that it may function unhindered. Once XAMPP has been installed, the program may be started using the “xampp-control” application which will be found in the xampp folder wherever it was installed on the computer. The control panel will look like the following:

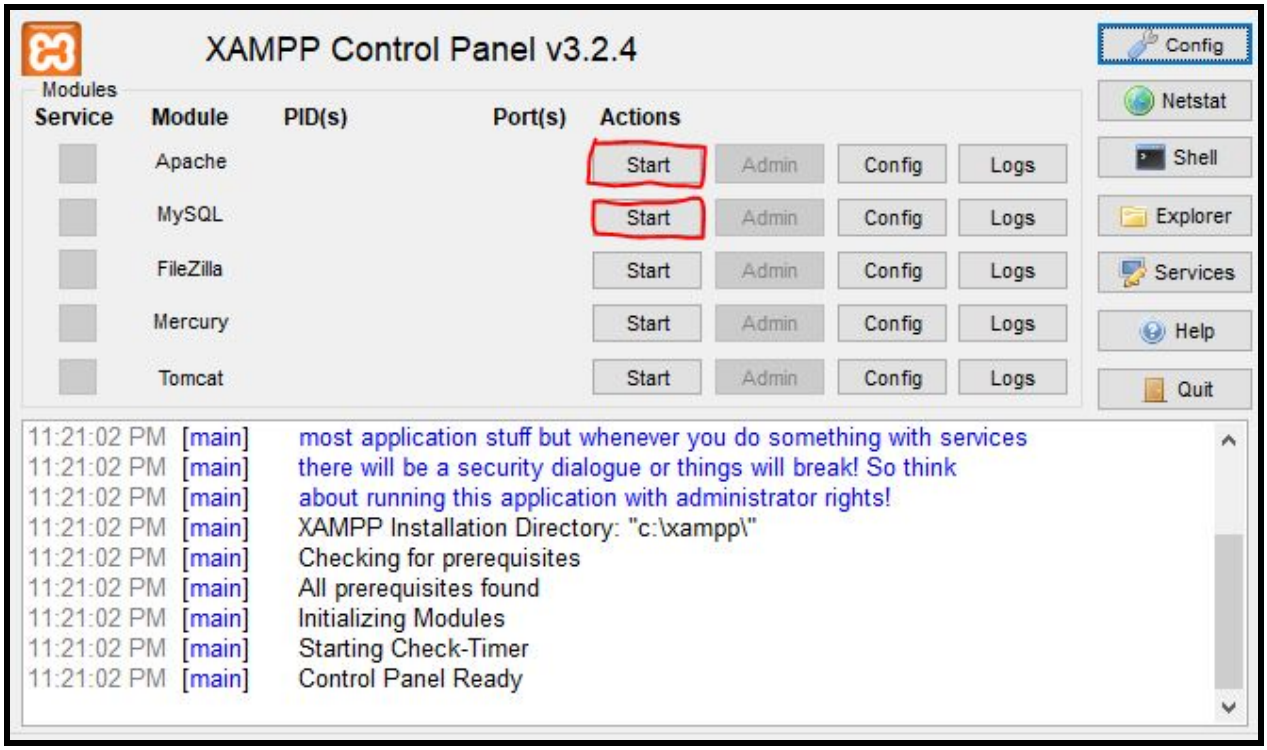


Figure 34: XAMPP Control Panel

Click the two buttons circled in red, these are the two modules that will need to run, the other three will not be used. Once both modules have been started in a successful manner, it will look like the following:

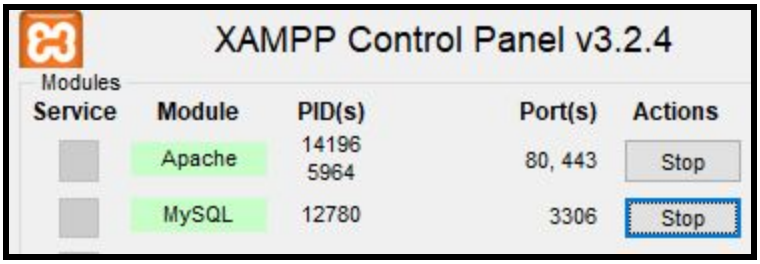


Figure 35: XAMPP Server Running

If there are any errors that show up during the start up process, try disabling all antivirus software and firewalls that may be active on this computer. This should allow the program to access any ports that it needs.

4.2.1 - Setting Up the PHP Script

While the MKR1000 handles sending the information to the server, the PHP script handles reading that information and putting it in the correct place. It is not recommended to modify the PHP script in any way, as it may cause issues with communicating to the database. The PHP script will be supplied both in the appendix of this report and from Dr. Farhad. It is important that the script is named SensorScript.php. This matches the address specified in the GET command, this is shown below:

```
//Create the message that gets sent to the SensorScript.php script
message = "GET /TireSensing/SensorScript.php?time=" + String(timeSe
```

Figure 36: GET Command to Match PHP Script

In addition to the name being correct, this file must be put in a very specific place. Specifically, it must be put in a folder called “TireSensing” that is then placed within the “htdocs” folder. The “htdocs” folder is within the XAMPP folder that is wherever the XAMPP program was installed on the computer. This is illustrated below:

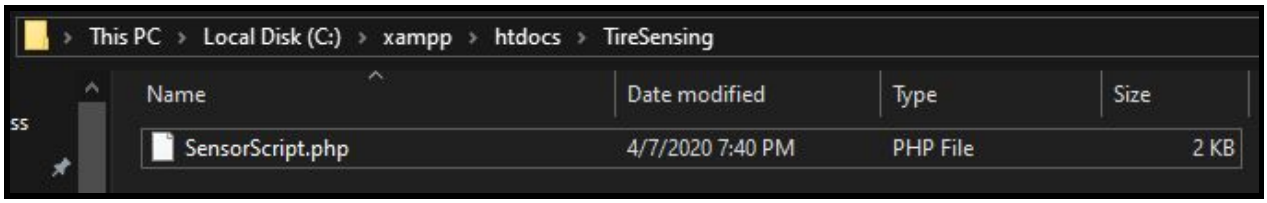


Figure 37: File Location for PHP Script

The “htdocs” folder will already exist inside the xampp folder that was created when XAMPP was installed. Note that it is important that case and spaces are important, each folder and file name must be exactly as shown above or it will not work.

4.3 - MySQL Database and its Setup

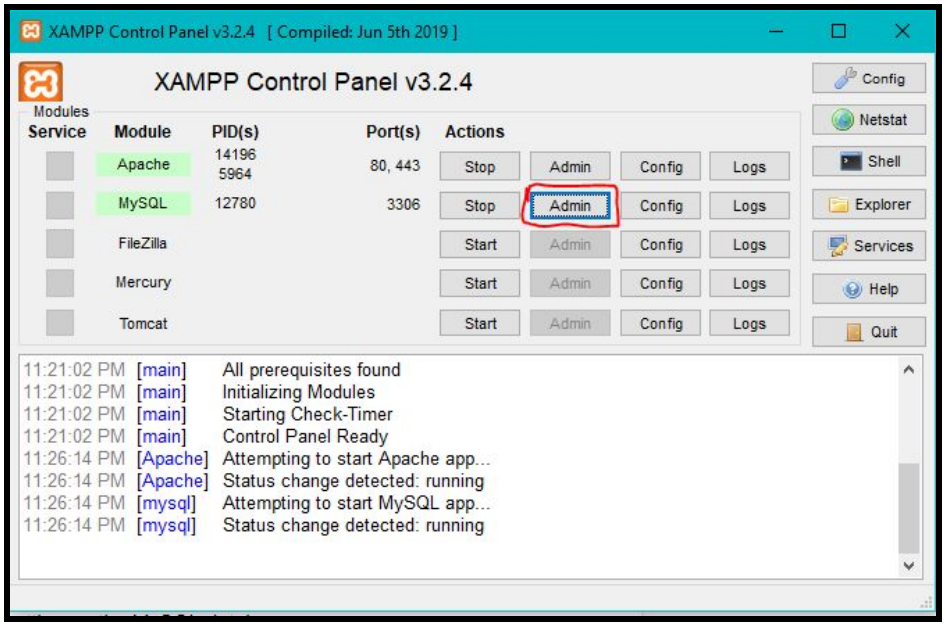
The final portion of setup necessary to get this sensing system to work on any computer and router is the setup of the MySQL Database. There are three parts of the MySQL database setup:

1. Database Name
2. Table Name
3. Column Names

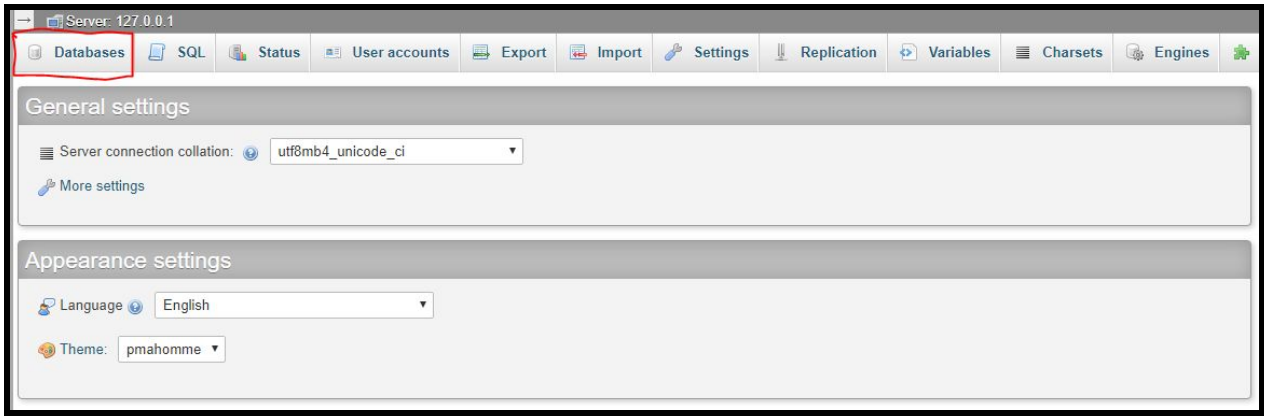
All three of these options need to be set up in the proper way so that it will match the PHP script that connects to it.

The following is a step by step guide for setting up the MySQL database.

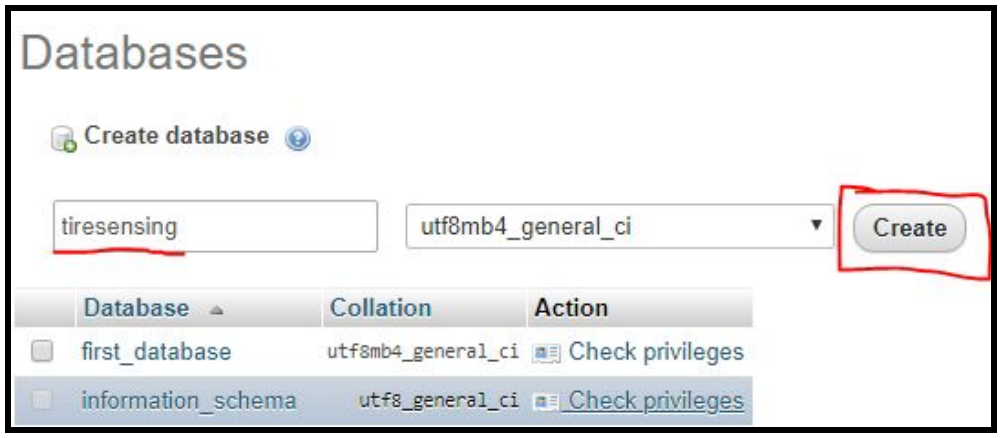
1. Open the admin portal to the MySQL database



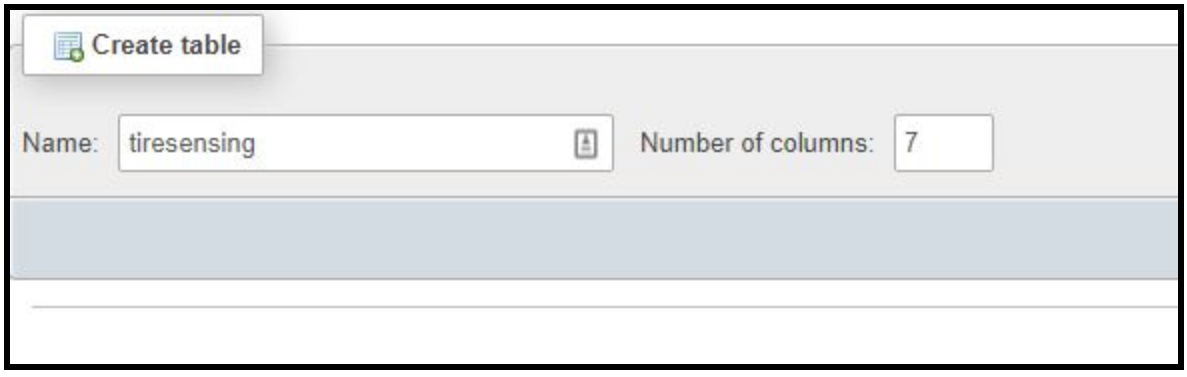
2. Access the databases tab



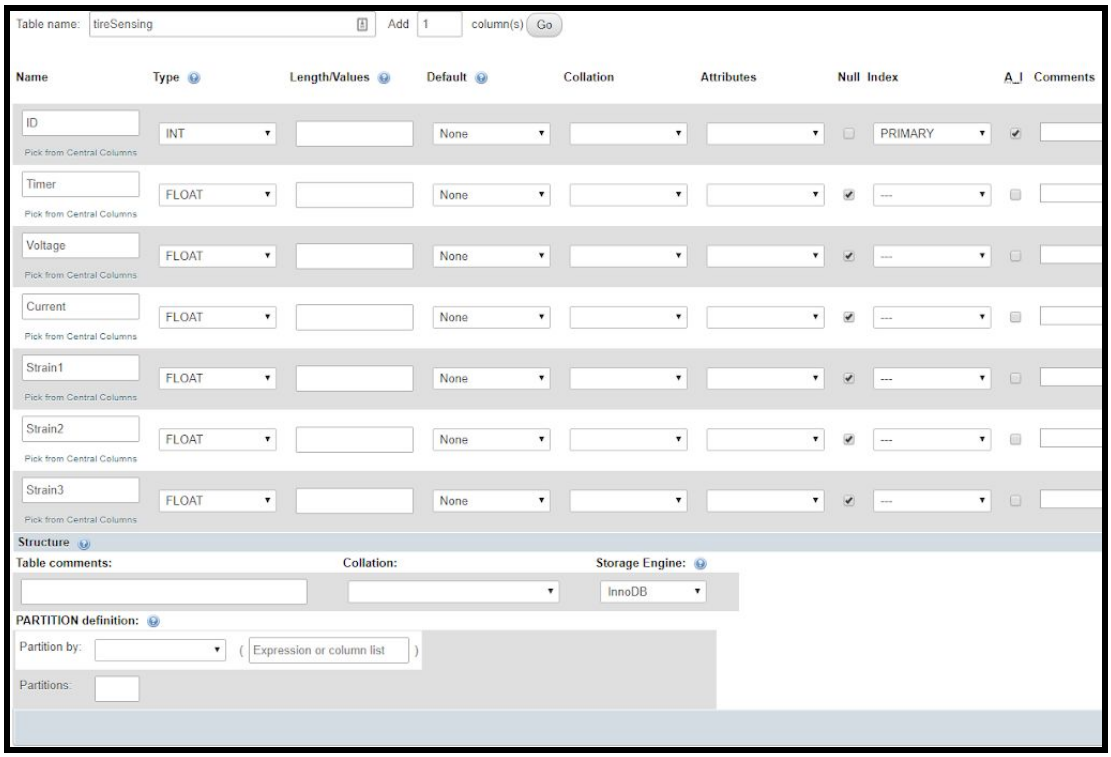
3. Create a new database named “tiresensing”. Ensure that the database name typed in matches exactly what is underlined below and in quotes in the previous line.



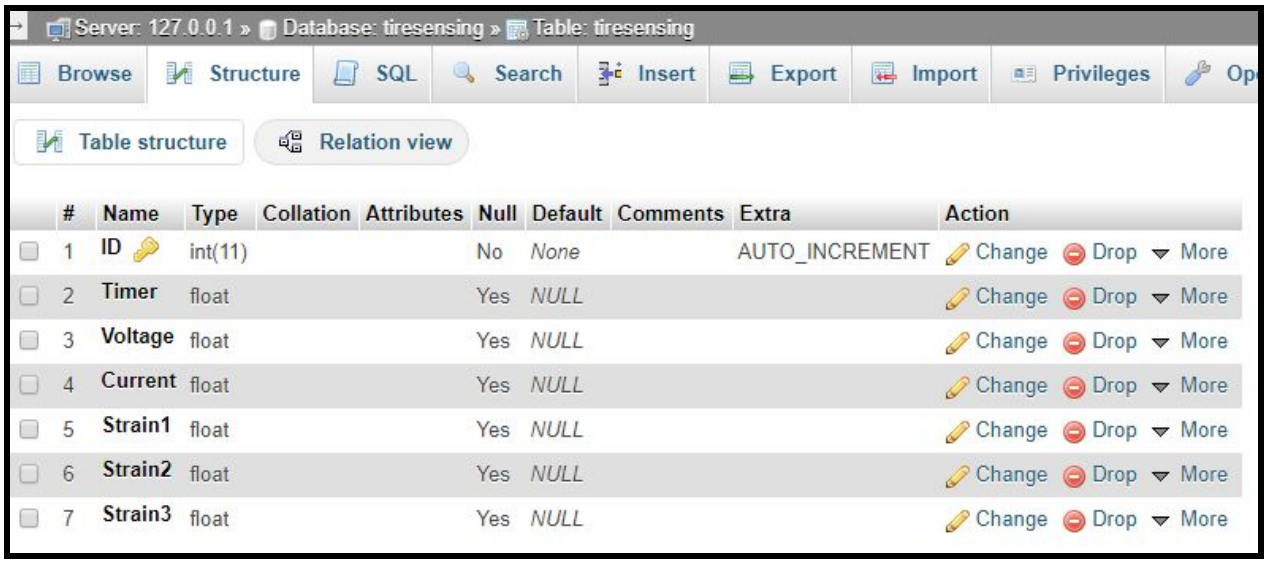
4. Create a new table, named “tiresensing” with 7 columns, click “Go” to create it.



5. Configure the columns the same as below:



6. Verify that the structure of this table is as shown below:



If all pieces parts of the above image match the database on the computer, the database is ready to be used.

4.4 - Running the Test

4.4.1 - Startup and Monitoring

Once the database has been configured as above, and the Arduino has had the updated .ino file (with new network parameters and server IP address) uploaded to it, testing the database and running the test is now possible. As stated previously it is recommended that the test is run on a wireless network provided by a simple wireless router such as the TP-Link AC1200 or any equivalent. Once the wireless network has been configured, update the MKR1000 code to match the new wireless network name and password. Additionally, once the laptop is connected to the wireless network it is necessary to find the ip address assigned to the laptop by the router. This can be done by viewing the router's configuration page, or by following this guide. Take that IP address that is found and change this in the beginning of the MKR1000's code to match the new IP address of the local computer.

Upload this code to the MKR1000 and let it run, if testing at the computer, simply plugging it in and leaving it plugged in to the USB cable will work. If running the test in the tire, upload the code to the MKR1000 from the computer, disconnect, and then plug the battery in. Ensure that on the local computer the XAMPP server has the Apache module and the MySQL module running, if they are not running, hit the circled buttons below:

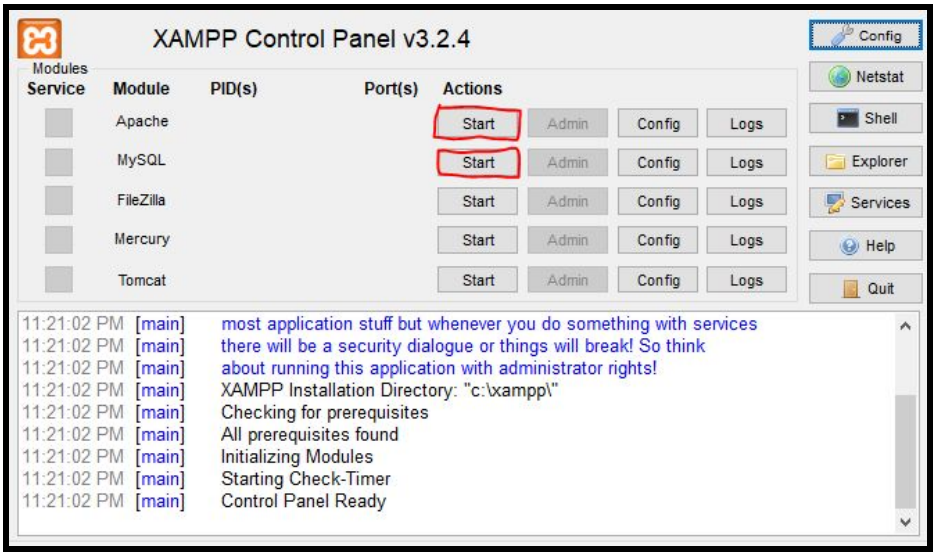


Figure 38: Buttons to start server and MySQL database

At this point, the MKR1000 will connect to the server and start sending sensor readings to the database once any time delays have been passed. This can be checked by clicking the “Config” button in the MySQL module. This will open the MySQL database configuration page. To get to the database you created, click on “Databases”, then click on the database “tiresensing”, then click on the table “tiresensing”. If there are values that have been recorded into the database they will be shown now, or if it is empty, it will just show the names of the columns that have not been filled in yet.

4.4.2 - Data Control During Testing

The MKR1000 simply runs on a time delay of taking and sending readings every 30 seconds, which equates to 2000 data points every minute. This can be a lot of data to look through, and could create a lot of data that is unwanted. Once the MKR1000 and the rest of the sensing system is on the wheel and the tire has been put back on, it may be desired to stop recording sensor data that the MKR1000 sends to minimize bad data being put into the database. This can be done by simply shutting down the MySQL module through the XAMPP

administrative window. When it is time to start accepting data again, just turn the MySQL module back on. The first set of readings may have some bad data in it, but it should then work well afterwards.

4.4.3 - Data Processing

Once the data has been recorded into the database, there are likely two things that will want to be done with it: delete it (bad data), or export it to Excel or some other software for post processing. Any statistical, graph, or filtering work to be done on the data is best done outside of this database and in software such as Microsoft Excel that is more suited for that kind of task.

Deleting data is relatively simple and can be done by checking off all rows on a page and hitting delete, or by using SQL delete commands on a range of ID's which can be explained here.

Exporting to Excel is also easy, in the top bar when viewing the tiresensing table, there is an "Export" button. Using this button and adjusting the parameters will allow the easy export of any data required.

4.4 - Possible Issues

Due to COVID-19 testing was unable to be completed with this new system. It would be prudent to run tests on the sensing system before actually testing at the Stark State facility to ensure that everything is working as expected. One possible issue may be excessive noise due to the MKR1000 or from the inductance created by the motor rotating the wheel. A solution to this would be to create a filter on the data that gets rid of any possible noise.

Another possible issue is the current sensor may not be properly calibrated. This was mentioned in Section 2.1.1.1 and the necessary steps to fix this will be recalibrating using vendor guides.

5 - Conclusion

This senior design project was continuing work and improvements on a Piezoelectric energy harvester [PEH] for a tire application. This work has been led by the Advanced Energy and Sensor Lab at the University of Akron and has had student assistance from two senior design groups. The group's focus involved making great improvements to the measurement system and method for testing and data verification. The measurement system sampling rate has been vastly increased and now reads five sensors. A new component was developed to assist with assembling the PEH along with a conducted COMSOL study to better understand this process. A new testing methodology was established, involving the creation of an enclosure to house the measurement system and a new ideology where the enclosure rests on the rim via hose clamps. All of these improvements could not be verified and confirmed through testing due to COVID-19. This report presents thorough documentation and transition material to ensure that future work can be easily resumed. Special thanks to the Industry Advisory Council of the Mechanical Engineering Department for financial support.

6 - References

[1] Aliniagerdroudbari, Haniph, et al. “A Piezoelectric Sandwich Structure for Harvesting Energy from Tire Strain to Power up Intelligent Tire Sensors.” *2019 IEEE Power and Energy Conference at Illinois (PECI)*, 2019, doi:10.1109/peci.2019.8698908.

[2] Yang, Z., Zhou, S., Zu, J., & Inman, D. (2018, April 10). High-Performance Piezoelectric Energy Harvesters and Their Applications. Retrieved April 28, 2020, from https://www.sciencedirect.com/science/article/pii/S2542435118301260

[3] Bowen, C. R., & Arafa, M. H. (2014, December 22). Energy Harvesting Technologies for Tire Pressure Monitoring Systems. Retrieved from https://onlinelibrary.wiley.com/doi/full/10.1002/aenm.201401787?casa_token=qSUN8sROITkAAAAA:cWm4Dw2X65kreD0gbQlLrQscrZwgVJvRPN_LKiIY50z22KW6Xbs4tTOyqY42Kx4z37yi2TreAbqVsg

[4] Almohatrish, Z., Condo, M., Eberly, J., Wilson, J., (2019, May 8). Piezoelectric Energy Harvester Design Project Progress Report.

[5] Embedotronics Technologies. (2019, June 2). Arduino Sending Sensor Data to MySQL Server (PHPMYADMIN). Retrieved from https://create.arduino.cc/projecthub/embedotronics-technologies/arduino-sending-sensor-data-to-mysql-server-phpmyadmin-a604d4

[6] Silva, Silva, A. L., Varanis, M., Mereles, A. G., Oliveira, C., & Balthazar, J. M. (2018, December 10). A study of strain and deformation measurement using the Arduino microcontroller and strain gauges devices. Retrieved from http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1806-11172019000300401

[7] "AR904 Wheel." American Racing, www.americanracing.com/product/wheels/ar904/?attribute_pa_finish=satin-black.

[8] "How to Read Wheel Markings." Oponeo.co.uk - Tyres and Wheels Online, 22 Apr. 2020, www.oponeo.co.uk/blog/how-to-read-the-wheel-markings.

[9] Esmaeeli, Roja, et al. "Design, modeling, and analysis of a high performance piezoelectric energy harvester for intelligent tires." *International Journal of Energy Research* 43.10 (2019): 5199-5212.

Appendix A - MKR1000 Code

```
#include <SPI.h>
#include <WiFi101.h>

//Created by Nathan Embaugh, Spring 2020
//Readme: This script is designed to be uploaded to an Arduino MKR1000 that will connect to a wifi network,
//       take sensor readings and then send them via TCP communications to a server that is running on a computer
//       on the same network. A detailed how-to in using and editing this software is listed in the design report.
//       NOTE: All Serial.print commands are used solely for debugging the software when it is connected by USB cable
//       to the computer. The Arduino will work normally if just plugged in with a battery and not connected to the PC.

//Wifi Connection
char ssid[] = "NathanE" ;
char pass[] = "Arduino1" ;
int status = WL_IDLE_STATUS;

//Setting up client parameters
IPAddress server(192, 168, 0, 103) ;
WiFiClient client;

//Time Variables
float timeSent ;
int interval = 5000 ; //Time between sensor readings in microseconds
int sampleSize = 1000 ; //Number of readings taken per sample
unsigned long timestamp [1000]; //Needs to be as big as the sample size
unsigned long previousTime = 0 ; //Used for determining when to take a reading
unsigned long currentTime = 0 ; //Used for recording time to send to database
unsigned long initialTime = 0 ; //Used to make each set of readings start at zero seconds

//Sensor Variables
float voltage [1000]; // Array to hold voltage values, needs to be as big as sample size
float current [1000]; // Array to hold current values, needs to be as big as sample size
float strain1 [1000]; // Array to hold strain1 values, needs to be as big as sample size
float strain2 [1000]; // Array to hold strain2 values, needs to be as big as sample size
float strain3 [1000]; // Array to hold strain3 values, needs to be as big as sample size
```

```
//Conversion Variables
float sensorToVoltage = (1.0 / (7.5 / (7.5 + 30.0)) * (3.3 / 4095.0)) ; //Used to
convert from digital to analog value
float sensorToCurrent = (3.3 / 4096.0) * (100.0 / 500.0) ; //Used to convert from
digital to analog value including sensitivity
float sensorToStrain1 = (3.3 / 4096.0) ; //Used to convert from digital to analog
value
float sensorToStrain2 = (3.3 / 4096.0) ; //Used to convert from digital to analog
value
float sensorToStrain3 = (3.3 / 4096.0) ; //Used to convert from digital to analog
value

//Database Variables
String message ; //Used to hold the entire GET command

void setup()
{
  //Initialize serial and wait for port to open:
  Serial.begin(9600);

  // check for the presence of the shield:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // don't continue:
    while (true);
  }

  // attempt to connect to WiFi network:
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(10000);
  }
  Serial.println("Connected to wifi");
  printWiFiStatus();

  Serial.println("\nStarting connection to server...");
  // if you get a connection, report back via serial:
  if (client.connect(server, 80)) {
    Serial.println("connected to server and ready to sense");
```

```
  }
  //delay(300000); //Delay first sensing by 5 minutes, to allow for tire to be put
on wheel and setup on road machine.

}
//--------------------------------------------------------------------------------



/* Infinite Loop */
void loop()
{


  //Read the sensors locally and save readings to Arduino
  read_sensors() ;

  //Create packets and send to server database
 Sending_To_Database();


  delay(30000); //delay so many milliseconds until next reading


}

void read_sensors()
{

  analogReadResolution(12) ; // Gives us the higher resolution of 0-4095
  if (client.connect(server, 80)) //ensures that the server is connected before
taking readings
  {
    Serial.println("Server on, reading sensors") ;
    for (int i = 0 ; i < sampleSize;)
    {

      currentTime = micros() ;


      if (i == 0)
      {
        initialTime = currentTime ; //Store the time at which the first measurement
is taken,
        //used to make time start at zero in data processing
      }
```

```
     if (currentTime - previousTime >= interval) //Used to take readings at the
interval, sets the sampling rate
      {
       timestamp[i] = currentTime ;
       previousTime = currentTime ;

       voltage[i] = analogRead(A0) ;
       current[i] = analogRead(A1) ;
       strain1[i] = analogRead(A2) ;
       strain2[i] = analogRead(A3) ;
       strain3[i] = analogRead(A4) ;

        i = i + 1 ;
      }
    }
  }
  else
  {
  Serial.println("Server turned off") ;
  }
}



void Sending_To_Database()   //CONNECTING WITH MYSQL
{
  Serial.println("In send to db") ;
  if (client.connect(server, 80))
  {
    Serial.print("Sending to database") ;
    for (int z = 0 ; z < sampleSize ; z++)
    {
     //Reestablish connection to database
     client.connect(server, 80) ;

     //Do Data modification, includes converting from digital sensor values to
actual voltage, current, and strain
      //as well as setting time to start at zero

     voltage[z] = voltage[z] * sensorToVoltage ;
     current[z] = voltage[z] * sensorToCurrent ;
     strain1[z] = voltage[z] * sensorToStrain1 ;
     strain2[z] = voltage[z] * sensorToStrain2 ;
     strain3[z] = voltage[z] * sensorToStrain3 ;

     timeSent = (timestamp[z] - initialTime) / 1000000.0 ; //Sets starting time to
```

```
zero, converts from microseconds to seconds

     //Create the message that gets sent to the SensorScript.php script
     message = "GET /TireSensing/SensorScript.php?time=" + String(timeSent, 3) +
"&voltage=" + String(voltage[z], 4) +
                "&current=" + String(current[z], 4) + "&strain1=" +
String(strain1[z], 4) + "&strain2=" +
                String(strain2[z], 4) + "&strain3=" + String(strain3[z], 4) +
                " HTTP/1.1\r\n" + ("Host: 192.168.0.101\r\n") + ("Connection:
close\r\n\r\n") ;

     //Send the message
     client.print(message) ;

     //Used with Serial Monitor to debug if necessary
     Serial.println(message) ;

    }

  }


  else
  {
    // if you didn't get a connection to the server:
   Serial.println("connection failed");
  }
}


void printWiFiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
 Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

Appendix B - PHP Script

```php
<?php
class dht11
{
 public $link='';
 function __construct($time,$voltage,$current,$strain1,$strain2,$strain3)
 {
  $this->connect();
  $this->storeInDB($time,$voltage,$current,$strain1,$strain2,$strain3);
 }

 function connect()
 {
  $this->link = mysqli_connect('localhost','root','') or die('Cannot connect to the DB');
  mysqli_select_db($this->link,'tiresensing') or die('Cannot select the DB');
 }

 function storeInDB($time, $voltage, $current,$strain1,$strain2,$strain3)
 {
  $query = "insert into tiresensing set Timer='".$time."', Voltage='".$voltage."',
  Current='".$current."', Strain1='".$strain1."', Strain2='".$strain2."',
  Strain3='".$strain3."'";
  $result = mysqli_query($this->link,$query) or die('Errant query:  '.$query);
 }

}
if($_GET['time'] != '' and  $_GET['voltage'] != '' and $_GET['current'] != '' and
$_GET['strain1'] != '' and $_GET['strain2'] != '' and $_GET['strain3'] != '')
{
 $dht11=new
 dht11($_GET['time'],$_GET['voltage'],$_GET['current'],$_GET['strain1'],$_GET['strain2'],$_GET['s
 train3']);
}

?>
```